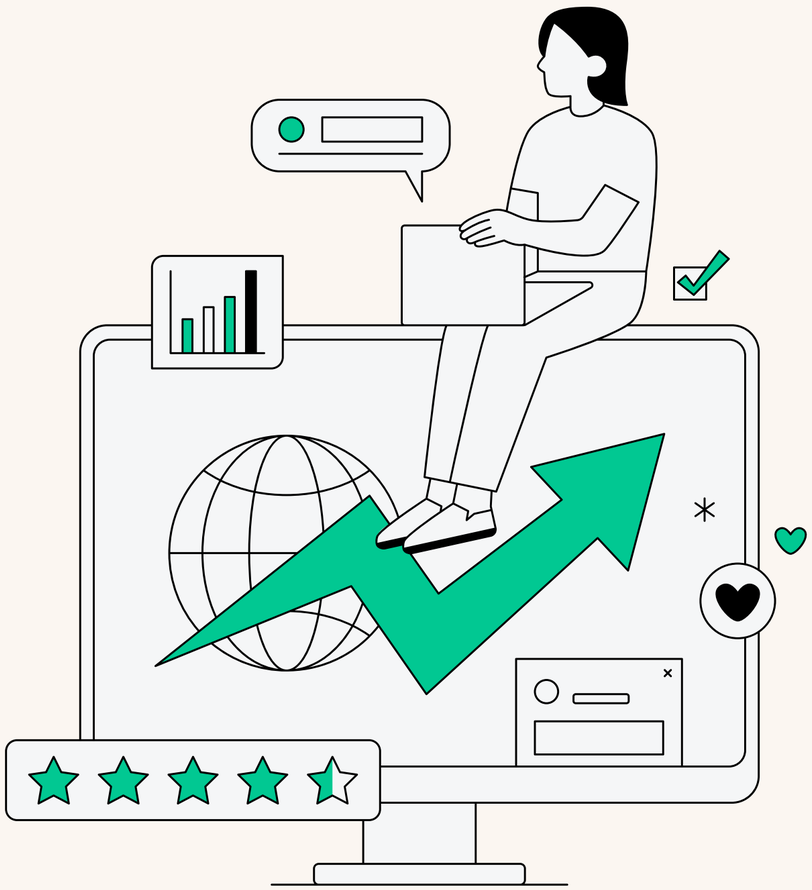
Presented by Chaahna Chandiramani

Serving HuggingFace Models in Production

A Scalable Sentiment Analysis API



Introduction & Model Selection

- Demonstration of Sentiment Analysis using a pre-trained Hugging Face model - distilbert-base-uncased-finetuned-sst-2-English
- Built for performance, scalability, and ease of deployment
- Includes server component, containerization, and parallel request testing

Why this model?

Balanced Simplicity and Relevance: Sentiment analysis is a widely understood NLP task with clear input-output behavior, making it ideal to demonstrate inference pipelines.

Speed and Lightweight Architecture: Built on **DistilBERT**, it's significantly faster and smaller than full BERT models while retaining over 95% of its performance which aligns well with container-based, scalable deployment.

General-Purpose Applications: Sentiment classification is a common NLP task across industries (e.g., analyzing product reviews, support tickets, or social media), making the demo relatable and practical.

System Components

01. **Hugging Face Model**
The sentiment analysis model is loaded at server startup.

02. **FastAPI App**
A lightweight web framework to define the API endpoint (/predict) and handle POST requests.

03. **Uvicorn Server**
ASGI server that runs the FastAPI app and supports asynchronous request handling.

04. **Docker Container**
Encapsulates the model, FastAPI app, and dependencies for consistent deployment across environments.

05. **Parallel Requests**
A Python notebook simulates multiple clients sending concurrent POST requests to test throughput.

main.py

C: > Users > chaah > Desktop > lemay.ai > task_2 > main.py > ...

```python
# main.py

from fastapi import FastAPI
from pydantic import BaseModel
from transformers import pipeline

# Initialize FastAPI app
app = FastAPI()

# Loads the sentiment analysis model once when the server starts
classifier = pipeline("sentiment-analysis", model="distilbert-base-uncased-finetuned-sst-2-english")

# Defines the input structure with Pydantic for data validation
class TextInput(BaseModel):
    text: str

# Defines the POST route
@app.post("/predict")
def predict_sentiment(data: TextInput):
    # Uses the classifier to analyze the text
    result = classifier(data.text)[0]
    # Returns the label and confidence score
    return {
        "label": result["label"],
        "score": round(result["score"], 4)
    }
```

Dockerfile

C: > Users > chaah > Desktop > lemay.ai > task_2 > Dockerfile

```
# Uses official Python base image
FROM python:3.10-slim

# Sets up a working directory in the container
WORKDIR /app

# Copying requirements
COPY requirements.txt .

# Installing dependencies
RUN pip install --no-cache-dir -r requirements.txt

# Copying the app code
COPY main.py .

# Exposes the port for FastAPI
EXPOSE 8000

# Starts the server using Uvicorn
CMD ["uvicorn", "main:app", "--host", "0.0.0.0", "--port", "8000"]
```

requirements.txt

```
fastapi
uvicorn[standard]
transformers
torch
```

.dockerignore

```
__pycache__/
venv/
*.pyc
*.pyo
*.pyd
```

Jupyter Notebook (parallel_requests_demo.ipynb)

In [1]:
```
import requests
import time
from concurrent.futures import ThreadPoolExecutor, as_completed
import pandas as pd

API_URL = "http://localhost:8000/predict"
```

In [2]:
```
texts = [
    "Today has been a lovely day!",
    "This is the worst experience ever.",
    "Not bad, but could be better.",
    "Absolutely fantastic!",
    "Terrible service and rude behavior.",
    "Everything was perfect.",
    "I'm not sure how I feel about this.",
    "Awful. Just awful.",
    "It was okay, nothing special.",
    "Best decision I made!"
]
```

In [3]:
```
def send_request(text):
    try:
        response = requests.post(API_URL, json={"text": text})
        return {
            "text": text,
            "status_code": response.status_code,
            "response": response.json()
        }
    except Exception as e:
        return {
            "text": text,
            "status_code": "Error",
            "response": str(e)
        }
```

In [4]:
```
start_time = time.time()
results = []

with ThreadPoolExecutor(max_workers=5) as executor:
    futures = [executor.submit(send_request, text) for text in texts]
    for future in as_completed(futures):
        results.append(future.result())

end_time = time.time()
duration = end_time - start_time
```

In [5]:
```
df = pd.DataFrame(results)
df["duration_secs"] = round(duration, 2)
df
```

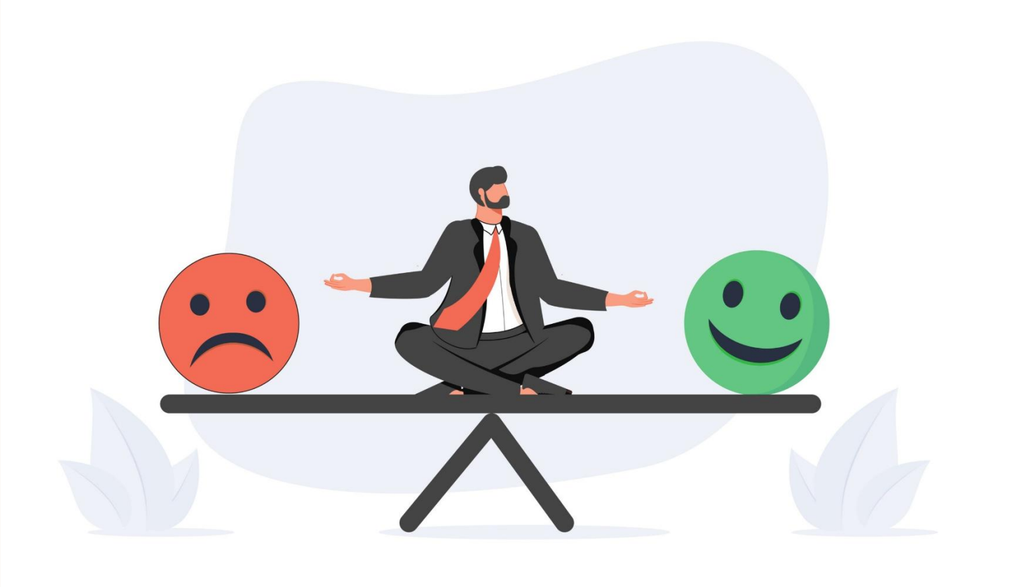
System Demonstration

Thank you

GitHub Repository - https://github.com/Chaahna/Sentiment-Analysis-API-with-HuggingFace

Future Scope

- Add GPU support and test with larger transformer models (e.g RoBERTa)
- Deploy to Kubernetes or AWS ECS for production scaling
- Expand to multi-language or emotion classification models
- Integrate logging, monitoring, and authentication for secure, robust deployments



Chaahna Chandiramani | Gmail | GitHub | LinkedIn | (778)-869-4628